Proceedings of the 2003 IEEE
International Symposium on Intelligent Control
Houston, Texas • October 5-8, 2003

On-line Nonparametric Regression to learn State-Dependent Disturbances

Bas J. de Kruif
University of Twente, Drebbel institute of Mechatronics
P.O. Box 217, 7500 AE, Enschede, The Netherlands
email:b.j.dekruif@utwente.nl (corresponding author)

Theo J.A. de Vries
University of Twente, Drebbel institute of Mechatronics
P.O. Box 217, 7500 AE, Enschede, The Netherlands

***Abstract*— A combination of recursive least squares and weighted least squares is made which can adapt its structure such that a relation between in- and output can be approximated, even when the structure of this relation is unknown beforehand. This method can adapt its structure on-line while it preserves information offered by previous samples, making it applicable in a control setting. This method has been tested with computer-generated data, and it is used in a simulation to learn the non-linear state-dependent effects, both with good success.**

***Keywords*— Function approximation, Least Squares, On-line structure adaption, Non-parametric regression**

I. Introduction

The use of past experience in control can increase the performance of the controlled system significantly by avoiding an error previously made. When a control system wants to 'learn' from its experience, it should be able to (compactly) store this experience such that it can use this experience to avoid the formerly made mistakes [1]. One possibility to store experience is with some kind of neural network that realises a mapping. The mapping summarises the examples by a function. For example, one might think of identifying the force that is required to compensate for the friction as function of the velocity in a mechanical setup. The neural network stores the mapping between velocity and force based on the examples. This mapping represents the experience of previously encountered friction forces and it can be used to decrease the tracking error due to the friction.

Roughly speaking, the learning of this mapping can be done in two ways: first collect all the samples, then determine the mapping; this is called *off-line learning*. The second way is called *on-line learning* and tries to find the mapping based on one sample a time.

In this paper, the on-line learning is considered for identifying and subsequently compensating for non-linear state dependent effects. We are interested in the on-line method due to the following advantages:

- Possibility to identify time variant functions.
- Learning in areas of the input domain not covered by the a-priori gathered examples.
- Continuous operation.

Although there are numerous methods for either adapting the parameters with only one example a time, e.g. [2], [3], or adapting the structure with the complete data set, e.g. [4]–[8], the combination of adapting the structure and the parameters with only one example a time is rather sparse [9]–[11]. However, this combination is useful in a learning control setting, because accurate information about the task is often unavailable beforehand. The methods mentioned in [11] are the so called 'lazy' approximators which require to store all the data and they make a model only when a prediction is required. The storage of all the data is not possible in a control setting because there is a continuous stream of data coming in. In [9] and previous work of these authors, the approximation is made by a set of local models. The validity of these models is determined by a weight matrix that is updated recursively. If a region of the input domain is not covered by the models, a new model is inserted. However, this method requires that some initial regions are constructed, while new ones can be included. Instead of dividing the input space in regions in which a local model is valid, one can identify a limited set of examples depending on an interpolation scheme that represent the data. With these samples a global model can be build.

The goal of this paper is to formulate an on-line approximator that construct a global model out of a limited set of selected examples. Such an approximator allows for both structural and parametric adaptations and can be used in a control setting.

This paper will start with some necessary background in section II on least squares and some variations of these. In section III various least squares methods are combined such that the method is capable of adapting its structure. An example will be presented in section IV to test if it is working. Furthermore, a simulation is performed in which the function approximator has to identify the non-linear state dependent effects and compensate for these to show that the approximator is capable of doing for which it was designed. In the last section conclusions are drawn.

II. Function Approximation

The goal of a function approximator is to approximate a relation between an input and an output based on a set of examples. Consider a given set of training samples $\{\mathbf{x}_k, y_k\}_{k=1\ldots N}$, in which $\mathbf{x}_k$ is the input vector and y_k is the corresponding target value for sample k. After training, a function approximator implements a mapping $\mathbf{x} \rightarrow \hat{y}(\mathbf{x})$ that

can be evaluated for any $\mathbf{x}$, i.e., it can be used to calculate an output for inputs that are not contained in the training set.

A. Off-line approximation

In the case of an off-line approximator, all the data is available to form the mapping. This in contrast to the on-line approximator for which the samples become present one at a time. The on-line case will be treated hereafter.

1) Least Squares: The least squares method searches for a linear relation between a set of indicators $f_k(\mathbf{x}), k = 1 \ldots n$ and the output $\hat{y}$:

$$\hat{y}(x) = b_1 f_1(\mathbf{x}) + b_2 f_2(\mathbf{x}) + \ldots + b_n f_n(\mathbf{x}) \tag{1}$$

In this equation, the elements of $\mathbf{b}$ are the n parameters that will get a value during the training by minimising the summed squared approximation error over all examples. The indicators can be non-linear functions of the input vectors. Define the matrix $\mathbf{X}$ and column-vector $\mathbf{y}$ containing respectively the indicators for all the k samples and the target values:

$$\mathbf{X} = \begin{pmatrix} f_1(\mathbf{x}_1) & f_2(\mathbf{x}_1) & \cdots & f_n(\mathbf{x}_1) \\ f_1(\mathbf{x}_2) & f_2(\mathbf{x}_2) & \cdots & f_n(\mathbf{x}_2) \\ \vdots & \vdots & \ddots & \vdots \\ f_1(\mathbf{x}_N) & f_2(\mathbf{x}_N) & \cdots & f_n(\mathbf{x}_N) \end{pmatrix} \tag{2}$$

$$\mathbf{y} = \begin{pmatrix} y_1 & y_2 & \cdots & y_N \end{pmatrix}^T \tag{3}$$

in which the subscripts of $\mathbf{x}$ and y denote the sample number.

The target vector $\mathbf{y}$ is assumed to be corrupted by independent and identically distributed (i.i.d.) noise, denoted by ϵ. Using the matrix notation introduced above, the following relation holds:

$$\mathbf{y} = \mathbf{X}\mathbf{b} + \epsilon \tag{4}$$

with, by the assumption on the noise,

$$E(\epsilon) = 0 \quad \text{and} \quad E(\epsilon\epsilon^T) = \sigma^2\mathbf{I} \tag{5}$$

The minimisation of summed squared approximation error is given as:

$$\min_{\mathbf{b}} \|\mathbf{X}\mathbf{b} - \mathbf{y}\|_2^2 \tag{6}$$

The values of $\mathbf{b}$ that minimise the above expression can be found by setting the derivatives to $\mathbf{b}$ equal to zero, resulting in the well known normal equations, see e.g. [2], [12], [13]:

$$\left(\mathbf{X}^T\mathbf{X}\right)\mathbf{b} = \mathbf{X}^T\mathbf{y} \tag{7}$$

2) Weighted Least Squares: In the case of ordinary least squares treated in the previous subsection, the examples are all weighted equally for the final approximation. However, if the variance of the additive noise is not equal for the examples, or if the noise is correlated between the samples, i.e. $E(\epsilon\epsilon^T) \neq \sigma^2\mathbf{I}$, then the use of the normal equations as stated in (7) will not give a minimal variance solution. The weighted least squares method is introduced to find a minimal variance solution when (5) does not hold. This method assigns more weight to samples of which the variance of the additive noise is smaller.

By use of a linear mapping, the variance on all the example samples can be made equal, so that the normal equations can be use to obtain a minimal variance solution. In the case of unequal variance, the relation between the targets and the indicators can still be given with $\mathbf{y} = \mathbf{X}\mathbf{b} + \epsilon$ with $E(\epsilon) = 0$. However, the variance is given as: $E(\epsilon\epsilon^T) = \sigma^2\mathbf{V}$. Because $\mathbf{V}$ is symmetric positive definite, there exists a Cholesky decomposition $\mathbf{P}\mathbf{P}^T$ with $\mathbf{P}$ lower diagonal such that $\mathbf{P}\mathbf{P}^T = \mathbf{V}$ [13]. Multiplying (4) from the left with $\mathbf{P}^{-1}$ results in:

$$\mathbf{P}^{-1}\mathbf{y} = \mathbf{P}^{-1}\mathbf{X}\mathbf{b} + \mathbf{P}^{-1}\epsilon \tag{8}$$

For this transformed representation the additive noise is given as:

$$\mathbf{f} = \mathbf{P}^{-1}\epsilon \quad \text{with} \quad E(\mathbf{f}) = 0 \tag{9}$$

The variance of the additive noise can now be calculated as:

$$\begin{aligned} E(\mathbf{f}\mathbf{f}^T) &= E(\mathbf{P}^{-1}\epsilon\epsilon^T\mathbf{P}^{-T}\sigma^2), \\ &= \mathbf{P}^{-1}E(\epsilon\epsilon^T)\mathbf{P}^{-T}\sigma^2, \\ &= \mathbf{P}^{-1}\mathbf{P}\mathbf{P}^T\mathbf{P}^{-T}\sigma^2, \\ &= \mathbf{I}\sigma^2. \end{aligned} \tag{10}$$

So, by pre-multiplying $\mathbf{X}\mathbf{b} = \mathbf{y}$ by $\mathbf{P}^{-1}$ the noise becomes uncorrelated and equal for all samples. The minimal variance solution of this transformed problem can be obtained by the ordinary least squares method. The minimisation problem is formulated as:

$$\min_{\mathbf{b}} \left\|\mathbf{P}^{-1}(\mathbf{X}\mathbf{b} - \mathbf{y})\right\|_2^2, \tag{11}$$

with the solution

$$\mathbf{X}^T\mathbf{V}^{-1}\mathbf{X}\mathbf{b} = \mathbf{X}^T\mathbf{V}^{-1}\mathbf{y}. \tag{12}$$

B. On-line approximation

The above treated methods use all the examples for the calculation of the parameter vector $\mathbf{b}$. Whenever the examples are offered one at a time, as often happens when approximating time series, the above mentioned method have to do all the calculations from scratch, whenever a new sample is offered. This is computational demanding and requires the storage of all the previous samples. Especially if the number of samples is large, the second argument makes this method unattractive. An on-line, or recursive, method does not require the complete data set for its calculations, but uses an update algorithm to modify its approximation each time a new sample becomes available, thus avoiding the storage of all the samples.

1) (Weighted) Recursive Least Squares: The normal equation can be easily rewritten to a recursive form. This update scheme is known as recursive least squares (RLS). Including a new sample $\mathbf{x}$, a row vector, with target γ and a variance ν into equation (12), results in:

$$\begin{aligned} \begin{pmatrix} \mathbf{X} \\ \mathbf{x} \end{pmatrix}^T \begin{pmatrix} \mathbf{V}^{-1} & 0 \\ 0 & \nu^{-1} \end{pmatrix} \begin{pmatrix} \mathbf{X} \\ \mathbf{x} \end{pmatrix} (\mathbf{b} + \Delta\mathbf{b}) = \\ \begin{pmatrix} \mathbf{X} \\ \mathbf{x} \end{pmatrix}^T \begin{pmatrix} \mathbf{V}^{-1} & 0 \\ 0 & \nu^{-1} \end{pmatrix} \begin{pmatrix} \mathbf{y} \\ \gamma \end{pmatrix}. \end{aligned} \tag{13}$$

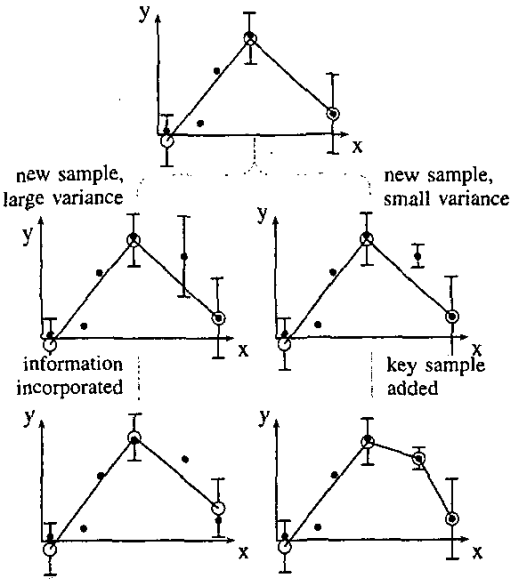

Fig. 1. Inclusion of a new sample. The offered sample can be incorporated into the existing data structure (left) or it can be used to extend the data structure (right)

The $\Delta \mathbf{b}$ can be calculated by some basic matrix calculations and the use of Woodbury's formula, see e.g. [2]. The resulting update law for the parameter vector $\mathbf{b}$ is given as

$$\mathbf{K} := \mathbf{K} - \frac{\nu \mathbf{K}\mathbf{x}^T\mathbf{x}\mathbf{K}}{1 + \nu \mathbf{x}\mathbf{K}\mathbf{x}^T} \quad (14)$$

$$\mathbf{b} := \mathbf{b} + \mathbf{K}\mathbf{x}^T\nu(\gamma - \mathbf{x}\mathbf{b}) \quad (15)$$

The matrix $\mathbf{K}$ is known as the covariance matrix and is related to Kalman filtering [14]. An important observation is that size of the covariance matrix and the parameter matrix $\mathbf{b}$ do not change because the number of indicator functions is kept equal.

III. Structure adaption

The functions $f_1(\mathbf{x})$ to $f_n(\mathbf{x})$ were given beforehand until now, and new indicators could not be included during the approximation of the data. This requires significant knowledge of the underlaying data structure or a very general set of indicators. The data structure is not always known beforehand and therefore it would be advantageous to have the ability to adapt this structure while the current approximation remains valid for the processed samples.

In this section the introduced least squares methods are combined such that the adaptation of the structure is made possible, without loss of previously stored information.

A. Principles

1) Using the variance: We would like to represent the processed samples with a small set of examples that represents all the training data obtained so far. This set of samples will be called the set of *key samples*. The clue for the method presented hereafter, is the fact that a key sample may represent numerous data points. This is incorporated in its uncertainty; when it represents many data points, the uncertainty will be small. This uncertainty is incorporated by the updates.

Whenever a new sample is offered to the approximator, two situations can occur:

- The information of the sample is represented by the existing key samples.
- The example should be used to expand the existing key samples.

Both these cases are illustrated in figure 1.

First consider the case where the information of the sample is incorporated into the existing data structure. This is given with the left path of figure 1. In this figure the top most graph is the original situation somewhere in the approximation process. In this graph the data is approximated by two straight lines that connect the three key samples. The key samples are denoted by the open circles. The training samples are represented by the solid dots, and the uncertainty is denoted with the error bars. The uncertainty of the key samples is influenced by the samples it represents. The right-most point has a uncertainty larger than the other key samples in this graph, because it only represents itself. In the middle graph a new sample is presented. The information of this sample will be incorporated into the existing data structure. Due to the inclusion of this information, the approximation will slightly change, as can be seen in the left bottom graph, and the uncertainty of the right most key sample is decreased. This variance is decreased, because the key sample does not represent only itself, but represents other samples as well.

Next to the inclusion of a sample into the existing data structure, it might be decided that the newly offered sample cannot be represented well enough by the present key samples and the new sample should become one of the key samples. This is illustrated by the right path of the figure. The uncertainty of the key samples and the new sample remain as they were.

2) Using the covariance: The inclusion of a new sample solely based on the uncertainty, given by the *variance*, will not give a correct result. Next to the variance of the key samples, there will also be *covariance* between the key samples that should be considered. This covariance originates from the fact that the key samples summarise a larger set of samples that make the key samples correlated; the key samples cannot be seen as independent observations. Because the key samples are coupled by the covariance, a shift in the target of a key sample will result in a shift of a target in the other key sample.

This coupling is illustrated in figure 2. When the target of the left key sample is increased due to some example, the target of the right key vector is decreased. The covariances between the vectors pose no problem for calculations because the WLS method uses a covariance matrix, which includes both variance and covariance.

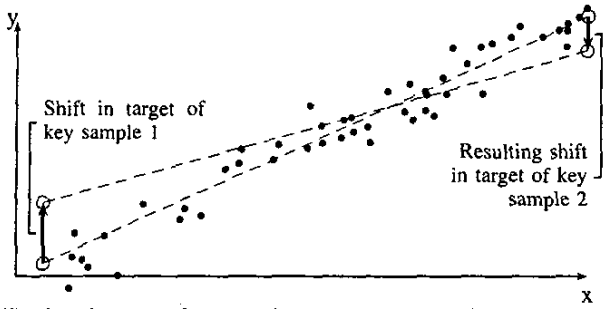

Fig. 2. Influence of the covariance on an approximation. Because the approximation of the left sample is increased, the approximation of the right sample is decreased.

B. Implementation

1) Structure of the indicators: In the figures, interpolation between the key samples was done with straight lines. Although this is a valuable option, it is not the only option. The indicator functions $f_i(\mathbf{x})$ that are used in the least squares setting can represent arbitrary functions, from which a linear combination should predict the targets as good as possible.

The method assumes that including a new sample as key sample into the data structure, will extend the set of functions that can be approximated and the new sample can be approximated with some accuracy. For example, it would be of no use to include more indicators of the form $f_i(\mathbf{x}) = \mathbf{x}$, because this will not extend the set of functions that can be approximated.

The set of indicators in which we are interested can be equalled to the kernel functions of the Support Vector Machines [15], [16]. This field offers indicators to approximate the data with e.g. splines, radial base functions, polynomials and so on.

2) Calculation of the covariance matrix: The ideas formulated in section III-A all use the target values and the covariance matrix of a set of key samples to represent the data. To summarise the complete data by a small subset of this data we have to equate the prediction and the variance on this prediction for the full data set and the reduced weighted data set. These two can be equated by selecting an appropriate covariance matrix $\mathbf{V}$ and target values for the WLS scheme.

By equating the *variance* of the prediction for both cases, the following equality is obtained:

$$\mathbf{x}^T(\mathbf{X}^T\mathbf{X})^{-1}\mathbf{x}\sigma^2 = \mathbf{x}^T(\mathbf{X}_{\mathbf{ks}}^T\mathbf{V}^{-1}\mathbf{X}_{\mathbf{ks}})^{-1}\mathbf{x}\sigma^2 \tag{16}$$

In this equation $\mathbf{X}_{\mathbf{ks}}$ is the indicator matrix of the form (3) in which the indicator function is evaluated at the key samples. This matrix is $n \times n$ in which n is the number of key samples. This equation gives us the means to calculate the matrix $\mathbf{V}$, such that the approximation variance of the full and reduced data set is equal. It follows that:

$$\mathbf{V}^{-1} = \mathbf{X}_{\mathbf{ks}}^{-T}\mathbf{X}^T\mathbf{X}\mathbf{X}_{\mathbf{ks}}^{-1} \tag{17}$$

which has a solution when $\mathbf{X}_{\mathbf{ks}}$ is not singular. Note that the matrix $\mathbf{X}^T\mathbf{X}$ is given in terms of the full data, but it can be calculated without storage of the full data. The size is $n \times n$ in which n is the number of indicators and can be updated when a new example becomes present as $\mathbf{X}^T\mathbf{X} \leftarrow \mathbf{X}^T\mathbf{X} + \mathbf{x}^T\mathbf{x}$.

Furthermore, also the targets of the key samples are required to summarise the data. The predicted values of the summarised and the full set should be equal. The prediction of the full set is given as:

$$\hat{y}(\mathbf{x}_{\mathbf{ks}}) = \mathbf{X}_{\mathbf{ks}}(\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{y} \tag{18}$$

The predicted value at the key samples as function of its targets is given as:

$$\begin{aligned} \hat{\mathbf{y}}_{\mathbf{ks}}(\mathbf{x}_{\mathbf{ks}}) &= \mathbf{X}_{\mathbf{ks}}(\mathbf{X}_{\mathbf{ks}}^T\mathbf{V}\mathbf{X}_{\mathbf{ks}})^{-1}\mathbf{X}_{\mathbf{ks}}^T\mathbf{V}\mathbf{y}_{\mathbf{ks}} \\ &= \mathbf{X}_{\mathbf{ks}}(\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}_{\mathbf{ks}}^T\mathbf{X}_{\mathbf{ks}}^{-T}\mathbf{X}^T\mathbf{X}\mathbf{X}_{\mathbf{ks}}^{-1}\mathbf{y}_{\mathbf{ks}} \\ &= \mathbf{y}_{\mathbf{ks}} \end{aligned} \tag{19}$$

In this equation the $\mathbf{y}_{\mathbf{ks}}$ are the targets of the key samples while the $\hat{\mathbf{y}}_{\mathbf{ks}}$ are the values of the predictions at the key samples. It is clear that by setting the targets of the key samples equal to the prediction of the full case, $\mathbf{y}_{\mathbf{ks}} = \hat{y}(\mathbf{x}_{\mathbf{ks}})$, the approximation will be equal.

3) Include the information: First the inclusion of the information of a sample into the existing data structure is treated. The inclusion can be done with the following steps:

1) Calculate $\mathbf{V}^{-1}$ with (17).
2) Include new sample $\mathbf{x}$ with target y and weight ν to the WLS problem as in (13) and solve.

Solving this WLS problem can be done non-recursively with the use of (12) while it can be done recursively with (14 and 15). In equation (14) it can be seen that the matrix $\mathbf{V}$ is not necessary to be explicitly calculated for an update. In the scheme above, it was only included for consistency with the ideas formulated in section III-A. The size of the matrix $\mathbf{X}_{\mathbf{ks}}$ does not change due to this update because no key vector is included. The same holds for the size of the matrix $\mathbf{X}^T\mathbf{X}$.

The inclusion of the information of a sample in the data representation is exact, meaning that the solution is identical to the solution that is obtained as if all samples are used in once.

4) Adapt the data structure: Next to the inclusion of the information of a sample into the existing data structure, the sample might give rise to a change of this data structure. The adaption of the data structure can be done by the following steps:

1) Calculate $\mathbf{V}^{-1}$ with (17).
2) Adapt $\mathbf{X}_{\mathbf{ks}}$ to include the new indicator. This new $\mathbf{X}_{\mathbf{ks}}$ has the form $\begin{pmatrix} \mathbf{X}_{\mathbf{ks}} & \mathbf{z} \\ \mathbf{x} & \chi \end{pmatrix}$.
3) Solve the WLS problem.

Due to the inclusion of a new key vector the size of $\mathbf{X}_{\mathbf{ks}}$ will increase. This will result in a modified version of (13):

$$\begin{aligned} &\begin{pmatrix} \mathbf{X}_{\mathbf{ks}} & \mathbf{z} \\ \mathbf{x} & \chi \end{pmatrix}^T \begin{pmatrix} \mathbf{V}^{-1} & \alpha \\ \alpha^T & \nu^{-1} \end{pmatrix} \begin{pmatrix} \mathbf{X}_{\mathbf{ks}} & \mathbf{z} \\ \mathbf{x} & \chi \end{pmatrix} (\mathbf{b} + \Delta\mathbf{b}) \\ &\quad = \begin{pmatrix} \mathbf{X}_{\mathbf{ks}} & \mathbf{z} \\ \mathbf{x} & \chi \end{pmatrix}^T \begin{pmatrix} \mathbf{V}^{-1} & \alpha \\ \alpha^T & \nu^{-1} \end{pmatrix} \begin{pmatrix} \mathbf{y} \\ \gamma \end{pmatrix}. \end{aligned} \tag{20}$$

This equation makes explicitly use of the weight matrix $\mathbf{V}$ for the update of the structure. The $\mathbf{z}$ is a column vector that gives the values of the new indicator, $\mathbf{z} = (f_{n+1}(\mathbf{x}_1) \quad f_{n+1}(\mathbf{x}_2) \quad \cdots \quad f_{n+1}(\mathbf{x}_N))^T$. The evaluation of this new indicator is made at the key samples. The χ is the value of the new indicator function at the new key sample: $f_{n+1}(\mathbf{x}_{\text{new}})$

Because the previous examples are omitted, the covariance of the new key sample with the other key samples cannot be calculated, which leaves α undetermined. This freedom can be used to include regularisation into the problem. By filling the values of this vector with some distance measure, the vector has a smoothing effect on the solution. A second option is to place zeros in this vector, with this it is assumed that the new key sample is uncorrelated to the other key samples. Because the true value of α is unknown, the inclusion is *not* exact as is the inclusion of the information into an existing data representation.

IV. Examples

In this section a computer-generated function will be approximated with the on-line approximator and a simulation result will be given where the approximator identifies and compensates for non-linear state dependent effects.

A. Computer generated function

The function that is to be approximated is given as

$$y(x) = \sin\left(\frac{1}{x + 0.06}\right) + \tanh\left(100\,(x - 0.5)\right) \qquad (21)$$

with $x_i \in (0, 1)$. This function consists of two parts, see figure 3: First, the left part of the summation that will start by fluctuation fast and which fluctuation will lessen, and secondly a steep function at $x = 0.5$. The first part is included to test if the number of key samples is indeed found to be larger for small values of x, which is required to approximate the data well. The second part is included to test if the method can approximate a near discontinuous function while it will not try to fit the noise.

The criterion to include or exclude an example as a key sample is not treated in this paper because of space limitations. Generally speaking, an example is included into the data structure if its target is too far off the prediction, considering the noise level and the variance of the approximation, while an example is excluded from the data structure if the introduced error when omitting it, is small relative to the noise level. The approximation error is allowed to be of the same magnitude as the noise level, to avoid overfitting [17].

The above mentioned function is approximated by splines that result in a piecewise linear approximation. The result is given in figure 3. In this figure the black line is the approximation, the gray dots the training data and the open circles are the key samples. It is clear that in the fast fluctuation areas more key samples are required. The approximation is not overfitting the data.

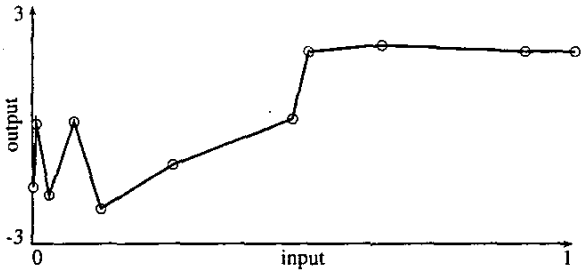

Fig. 3. Approximation of data by piecewise linear function.

B. Learning state dependent disturbances

Non-linear state dependent dynamics can be learnt and compensated for in a Learning FeedForward Control setting (LFFC) [18]–[21] which is based on Feedback Error Learning [22]. This method tries to identify the state dependent effects as function of their states and uses the approximation to compensate for these by feedforward. Because the effects are identified as function of their state, they can be compensated for even if the requested motion is non-periodic. One can think of identifying the friction force as function of the velocity and use this approximation to nullify the effect it has on the tracking error. The LFFC-scheme offers examples to the learning mechanism to approximate at each time step, and requires an estimation of the force to nullify the effect each time step. For the interested reader we refer to the above mentioned literature.

The model that is used in the simulation represents a linear motor. This motor can be described by the following differential equation:

$$m\ddot{x} + F_{\text{fric}}(\dot{x}) + F_{\text{cogg}}(x) = F_{\text{fb}} + F_{\text{ff}} \qquad (22)$$

In this equation m is the (unknown) mass, x is the position, $F_{\text{fric}}(\dot{x})$ is the friction force depending on the velocity, $F_{\text{cogg}}(x)$ is the cogging force depending on the position, F_{fb} is the applied force due to feedback and F_{ff} is the force due to the learnt feedforward. A PD-controller is used in the simulations to give a stable behaviour of the system.

The learning mechanism in the LFFC setting is used to compensate for the cogging, friction and the unknown mass. The mapping the approximator has to find, is a mapping from $(\ddot{x}, \dot{x}, x)$ to the feedforward force F_{ff} that will compensate for the above mentioned effects. The set of mappings from which the approximator searches is the set of piecewise linear functions. The reference position that the linear motor had to follow was a concatenation of (reproducible) arbitrary third order movements.

The tracking error of this method is depicted in figure 4. It can be seen that the tracking error is decreased significantly by the LFFC setting. The RMS value of the tracking error without learning is 98 [μm] for the last 30 [s] while the RMS of the tracking error in the learning case was 4.7 [μm]. Based on this, it can be concluded that the approximator was capable of finding a good approximation of the state dependent effects.

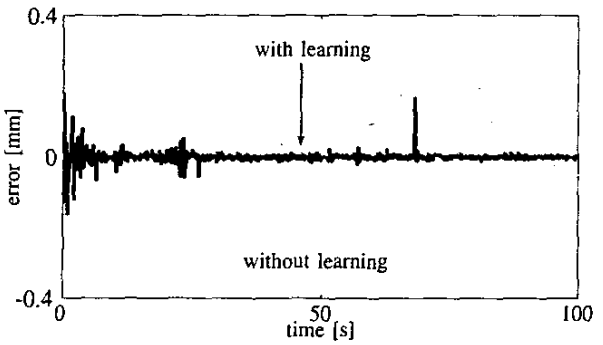

Fig. 4. Tracking error without (gray) and with learning (black)

At $t \approx 70$ [s] it can be observed that the tracking error is suddenly rather large. This can be explained because at this location the motor moves in a region that has not been covered before and the learning controller does not know what the feedforward forces should be.

In this simulation the convergence of the approximation is rather fast. The calculation time of the algorithm is still too high to run real time with a sample frequency of 1 [kHz], but it is believed that with a optimised implementation this simulation might run at a reasonable sample frequency. The simulation in which the dynamics of the system were calculated together with the on-line adaption of the function approximator took ca. 300 [s] at an AMD Athlon 1800+ personal computer. The final approximation required approximately 400 key samples. This result is better than the results obtained in [20]

V. Conclusion

In this paper a combination of recursive least squares and weighted least squares is presented. This combination can approximate a set of data by selecting training samples, the key samples, that can represent all the examples. This is done in an on-line fashion, making it applicable for learning in a control setting.

The representation not only uses the key samples, but it also assigns a weight to these samples. This weight allows for recursive parameter adjustment as well as structural changes, without loss of previous information.

The approximation is used in a simulation in which the state-dependent non-linear effects of a linear motor are identified and compensated for. The simulation has shown that this method works well with the learning feedforward control setting to nullify the disturbing effects. With the current implementation, the computation time is still too large to be used in a real time setting.

References

[1] P. J. Antsaklis, "Guest editor's introduction," *IEEE Control Systems*, vol. 15, no. 3, pp. 5–7, June 1995, special Issue on 'Intelligence and Learning'.

[2] Å. Björck, *Numerical Methods for Least Squares Problems*. Philadelphia: Society for Industrial and Applied Mathematics (SIAM), 1996.

[3] S. Haykin, *Neural Networks, A Comprehensive Foundation*. Upper Saddle River, New Jersey: Prentice Hall, Inc., 1994.

[4] S. Chen, F. N. Cowan, and P. M. Grant, "Orthogonal least squares learning algorithm for radial basis function networks," *IEEE Transactions on Neural Networks*, vol. 2, no. 2, pp. 302–309, March 1991.

[5] M. H. Fun and M. T. Hagan, "Recursive orthogonal least squares learning with automatic weight selection for gaussian neural networks," in *International Joint Conference on Neural Networks*, Washington, July, 1999.

[6] M. Martin, "On-line support vector machines for function approximation," Universitat Politècnica de Catalunya, Departament de Llenguatges i Sistemes Informàtics, Tech. Rep. LSI-02-11-R, 2002.

[7] B. J. de Kruif and T. J. A. de Vries, "Support-vector-based least squares for learning non-linear dynamics," in *Proceedings of the CDC'02*, 2002.

[8] A. Esposito, M. Marinaro, and S. Scarpetta, "An incremental local radial basis function network," in *ESANN'1998 Proceedings - European Symposium on Artificial Neural Networks Bruges (Belgium)*. D-Facto public., April 1998, pp. 21–26.

[9] S. Schaal, C. G. Atkeson, and S. Vijayakumar, "Scalable techniques from nonparameteric statistics for real-time robot learning," *Applied Intelligence*, in press.

[10] S. Schaal and C. G. Atkeson, "Constructive incremental learning from only local information." *Neural Computation*, vol. 10, no. 8, pp. 2047–2084, 1998.

[11] I. Uysal and H. A. Güvenir, "An overview of regression techniques for knowledge discovery," *The Knowledge Engineering Review*, vol. 14, no. 4, pp. 319–340, 1999.

[12] N. R. Draper and H. Smith, *Applied Regression Analysis*, 3rd ed., ser. Wiley Series in Probability and Statistics. New York: John Wiley & Sons, Inc., 1998.

[13] G. W. Stewart, *Matrix Algorithms*. Philidelphia: SIAM, 1998, vol. 1 : Basic Decompositions.

[14] L. Ljung, *System Identification, Theory for the user*, 2nd ed., ser. PTR Prentice-Hall information and system sciences series. Upper Saddle River, New Jersey: Prentice-Hall PTR, 1999.

[15] V. N. Vapnik, *The Nature of Statistical Learning Theory*, 2nd ed. New York: Springer-Verlag, 2000.

[16] N. Cristianini and J. Shawe-Taylor, *An Introduction to Support Vector Machines and other kernel-base learning methods*. Cambridge, UK: Cambridge University Press, 2000.

[17] S. A. Solla and E. Levin, "Learning in linear neural networks: The validity of the annealed approximation," *Physical Review A*, vol. 46, no. 4, pp. 2124–2130, August 1992.

[18] G. Otten, T. J. A. de Vries, J. van Amerongen, A. M. Rankers, and E. W. Gaal, "Linear motor motion control using a learning feedforward controller," *IEEE/ASME Trans. Mechatronics*, vol. 2, no. 3, pp. 179–187, 1997.

[19] J. G. Starrenburg, W. T. C. van Luenen, W. Oelen, and J. van Amerongen, "Learning feedforward controller for a mobile robot vehicle," *Control Engineering Practice*, vol. 14, no. 9, pp. 1221–1230, 1996.

[20] T. J. A. de Vries, W. J. R. Velthuis, and L. J. Idema, "Application of parsimonious learning feedforward control to mechatronic systems," *IEE Proc. D: Control Theory & Applications*, vol. 148, no. 4, pp. 318–322, July 2001.

[21] W. J. R. Velthuis, T. J. A. de Vries, and J. van Amerongen, "Learning feed forward control of a flexible beam," in *Proceedings of the 1996 IEEE International Symposium on Intelligent Control*, Dearborn, MI, 1996, pp. 103–108.

[22] M. Kawato, K. Furukawa, and R. Suzuki, "A hierarchical neural network model for control and learning of voluntary movement," *Biological Cybernetics*, vol. 57, pp. 169–185, 1987.